**2020-2021 Algos Exam**

\*Some proposed solutions may have errors, please comment / correct if so

**Question 1**

1. i)

Heap looks like:

20

6 15

4 1 10

In array form: 20 6 15 4 1 10

ii)

Heap looks like:

15

6 10

4 1

In array form: 15 6 10 4 1

1. i) Worse case input is when T has height N (adding elements in order ascending / descending order). x has value that is the max / min so need to traverse through entire tree

T(N) = theta(1) if N=1

T(N) = T(N-1) + d if N>1

ii) \*same as p29 in notes

T(N-1) = T(N-2) + 2d if N>2

T(N) = T(N-i) + id if N>1, let i=N-1

T(N) = T(1) + (N-1)d

T(N) = O(N)

1. (\*Probably wrong)

|  |  |  |  |
| --- | --- | --- | --- |
| N | binary | cost | total cost |
| 0 | 0000 | - | - |
| 1 | 1000 | 1 flip + 2^0 transmit = 2 | 2 |
| 2 | 0100 | 2 flip + 2^1 transmit = 4 | 6 |
| 3 | 1100 | 1 flip = 1 | 7 |
| 4 | 0010 | 3 flip + 2^2 transmit = 7 | 14 |
| 5 | 1010 | 1 flip = 1 | 15 |
| 6 | 0110 | 2 flip = 2 | 17 |
| 7 | 1110 | 1 flip = 1 | 18 |
| 8 | 0001 | 4 flip + 2^3 transmit = 12 | 30 |

Can choose a=4 to satisfy table

* Payment of 1 to flip
* Prepayment of 1 to flip back in future
* Prepayment/payment of 2 for transmission / future transmission of 2^j chars

Thus should take amortised O(1) time ???

**Question 2**

1. Some similarity to the quicksort partition algo?

i) Find element in indices 0 - (N-2) that is greater than k=20. Swap this with last element (N-1) and then call NewPartition on A[N-1]. Array = [29 21 7 6 13 50]. Swapped 50 with last element 21, taken theta(1) time as first element was greater than k=20. Needs to continue iterating through

ii)

~~procedure NewPartition(A,N,k,start)~~

~~If N>1 then~~

~~i = start~~

~~While(A[i] < k) //iterate thru and find first element > k~~

~~i++~~

~~temp=A[i]~~

~~A[i]=A[N-1]~~

~~A[N-1]=temp //swap with last element~~

~~If(A[i] < k) //if swapped element is less than k, can start from next element~~

~~NewPartition(A,N-1,k,i+1)~~

~~else //otherwise iterate from original element~~

~~NewPartition(A,N-1,k,i)~~

**Recursive (think this meets the requirements and works, lmk if doesnt**:

Procedure NewPartition(A,N,K):

If N>1:

if A[N]<k:

swap(A[1], A[N])

NewPartition(A[2:N], N-1, k)

else:

NewPartition(A[1:N-1], N-1 ,k)

End

Non-recursive solution:

(two pointers on either end of the array, incrementally checking if A[L] >= k and if it is, swapping it. Moving the pointers each time is what makes the unpartitioned subarray smaller by 1 with every loop)

Procedure newPartition(A, N, k):

L = 0

R = N-1

While (L < R) {

If (A[L] >= k) {

temp = A[L]

A[L] = A[R]

A[R] = temp

R = R – 1

}

Else {

L = L + 1

}

}

1. i) E[i] = I1 \* P(I = I1) = N \* k-1/M

Expected swaps = Probability of value being less than k = k-1/M (1-M range of values) \* number of vals N

ii) ~~M/M \* N = N ?~~

~~Let K = rv representing the chosen k. We partition on values of K.~~

iii) Using master method:

From equation we know that a=2,b=3. Thus N^log3(2) = roughly N^0.63

We know that in the *worst case*, f(N) should be N swaps (from ii) \* theta(1) time (swapping 2 elements), thus f(N) = O(N). And in the *best case*, no swaps occur thus f(N) = O(1).

In the worst case, by case 3 of the master method (N^logba is polynomially less than f(N)): T(N) = O(N)

In the best case, by case 1 of the master method (f(N) < N^logba): T(N) = Ω(N^log3(2))

**Question 3**

1. i) MST edges to be added:

Edge, weight

16,1

01,2

25,4

23,6

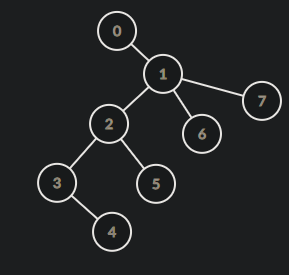
12,8

34,10

17,12

Total weight = 43

ii) NA

Not 100%  


b) (\*Probably wrong)

i) C(L,R) = 0 if L-R<=1 (cannot cut up any further thus no cost)

C(L,R) = min P[N-i] + C(L+i,R) or min P[N-i] + C(L,R-i)

Got something different //+1  
C(L,R) = min {lm P[R]-P[L]}, for i = L+1..R-1 if L-R>1  
C(L,R) = 0 if L-R<=1

ii) Bottom up approach (\*work in progress – other ideas welcome)

Procedure barcut(P,N)

C[0]=N

For i=0 to N:

Opt = N

For j=1 to i:

cost = (N-i) + C[i-j]

opt = min(opt,cost)

Endfor

C[i] = opt

Endfor

Return C[N]

End procedure

I think it’s easiest to think about in a recu sive manner first. Not 100% on the below as I haven’t actually implemented it but I think it makes sense.

procedure MinRodCutRecursive(P, L, R)

if (R – L <= 1) then

return 0

endif

opt <-

for k <- 1 to N-1

if P[k] > L and P[k] < R then

opt <- min(opt, R – L + MinRodCutRecursive(P, L, P[k])

+ MinRodCutRecursive(P, P[K], R))

endif

Endfor

if opt = then

opt <- 0

Endif

return opt

endprocedure

Then we can introduce some dynamic programming to save the calls to MinRodCutRecursive.

procedure MinRodCutDP(P, L, R)

for i <- L to R

for j <- L to R

C[i][j] <-

endfor

endfor

return MinRodCutDPAux(P, L, R, C)

endprocedure

procedure MinRodCutDPAux(P, L, R, C)

if (R – L <= 1) then

return 0

endif

if C[L][R] < then

return C[L][R]

endif

for k <- 1 to N-1

if P[k] > L and P[k] < R then

C[L][R] <- min(opt, R – L + MinRodCutDPAux(P, L, P[k], C)

+ MinRodCutDPAux(P, P[K], R, C)

)

endif

endfor

if C[L][R] = then

C[L][R] <- 0

endif

return C[L][R]

endprocedure